Integrating Power Grid Topology in Graph Neural Networks for Power Flow

M.G.A. Fatah[*]
Sustainable Energy Technology
Technische Universiteit Eindhoven

Bert J. Claessens[†]
Electrical Engineering
Technische Universiteit Eindhoven

Maarten Schoukens[‡]
Electrical Engineering
Technische Universiteit Eindhoven

Abstract

Recent advances in Neural Network offer an interesting opportunity to integrate graph topology in a Neural Network system. This framework is called Graph Neural Network (GNN). In power systems, an electrical power grid can be represented as a graph with high dimensional features and interdependency among buses. This perspective may offer a better state of the art machine learning for power systems analysis. This study seeks the opportunity to integrate power grid topology in the GNN framework for power flow application. A comparison between several GNN architectures with equivalent model complexities are discussed. The comparison is also done for various dataset sizes. The performance of GNN compared to fully connected Neural Network over different sizes of dataset is concluded.

***Index Terms*–Neural Network, Graph, Graph Neural Network, Power Flow**

1 Introduction

Our energy system is exposed to a range of trends that present both risks and opportunities in terms of managing our power system. On one hand, there is an increase in consumption due to electrification of heating and transportation. On the other hand, there is a rise of data and flexibility due to devices becoming connected and controllable. Harvesting this flexibility to benefit our society needs to happen in a grid secure way. It requires having visibility on the grid status and connectivity which can be done by utilizing the data from residential assets such as power and voltage [10].

Power flow or load flow is a calculation of buses' variables in an electrical power grid to identify the state of the grid at a particular load case [7]. One objective of power flow is to find out how much power needs to be produced by the generators within the power grid to satisfy a given demand. Although this problem is the core of electrical grid operation, the non-linear nature of electrical signals makes it difficult to solve [5].

Power flow calculation implements an iterative method. Calculating the power flow fully in AC signals is known as AC Power Flow (ACPF). Since AC values involve non-linear (sinusoidal) data, it relatively takes a longer time for the iterative method to finish the calculation, while electrical providers need to finish the calculation quickly for field related problems. A faster method to solve power flow calculation is using DC power flow (DCPF). DCPF uses small angle approximations that simplify ACPF non-linear complexity [9]. As a simplifying method, DCPF is not as accurate as ACPF but runs faster. However, the small angle approach fails for heavily-loaded networks, as the voltage angle differences become large [8]. Electrical providers are able to use DCPF as a warm start up for ACPF [11]. A more advanced power flow is AC Optimal Power Flow (ACOPF) which finds the optimal power that the generators have to produce to satisfy the demand [7]. The optimality is evaluated with regard to the cost that each generator incurs to generate the required power.

Involving sinusoidal AC signal dynamics results in a non-convex problem in the ACOPF [19] [23]. There have been attempts to utilize various methods to solve the problem. Recently, driven by the potential of accurately producing large amounts of data, machine learning has been considered as a new solution to solve this problem. For instance, a Neural Network model can be employed to predict the unknown variables of power flow by harnessing the known variables to emulate power flow calculation. The work in [13] uses a Fully Connected Neural Network (FCNN) to imitate the output of ACPF. However, because it makes use of information from nodes that are not adjacent in the network, the solution is not local. Moreover, FCNNs tend to overfit.

Latest developments in Neural Network offer a way to integrate graph topology in a Neural Network architecture [21] [32]. A new solution to solve ACPF is using Graph Neural Network (GNN). GNN exploits the structure of the graph data that is necessary for a local and scalable solution. The work in [25] implements GNN to solve the ACOPF problem. The author compares the result of several NN-based models on ACOPF and concludes that the result of the GNN model outperforms the FCNN model. However, the experiment setup in [25] does not explicitly state that the compared models adhere to equivalent complexities. Moreover, the comparison is done for a fixed size of dataset.

In this work, the use of GNN for power flow is investigated by comparing several model architectures with equivalent complexities i.e. same number of model parameters. This setup is arranged to see the performance of GNN compared to FCNN at an equal level. Moreover, the comparisons are done repeatedly for several different dataset sizes. This is done to check whether there are changes of the model performance with respect to the dataset size.

This report is organized as follows. Section 2 explains about graph data structure. Section 3 describes a short concept of Neural Network and its several classes namely Fully Connected Neural Network, Convolutional Neural Network, and Graph Neural Network. This section also elaborates the latter framework. Section 4 presents the concept of power flow in an electrical grid. Section 5 explains how the experiment is conducted. Section 6 discusses the result and analysis. Section 7 summarizes the conclusions. Section 8 gives recommendations for further works.

2 Graph

Graph is a structure consisting of objects and their connectivity. A graph G can be written as G = (V, E), where V is a set of nodes/vertices and E is a set of edges (nodes' connectivity). In the graph, v_i is the ith node and e_{ij} is the edge from the ith node to the jth node. A nodal feature matrix is denoted by X of n×f size. An adjacency matrix A is a matrix of n×n size where a_{ij} is 1 if $e_{ij} \in E$ and a_{ij} is 0 if $e_{ij} \notin E$. A degree matrix D shows the number of neighbors had by each node which is assigned diagonally. Take a look at figure 1 showing a simple graph with 3 nodes and 2 edges, with each node having 3 nodal features. The nodal feature matrix X, adjacency matrix A, and degree matrix D are shown in the equations.

Graph is basically how the world represents itself. Many naturally generated data are in the form of graphs. Proteins, social networks, and electrical power grids are all graph-shaped as shown in figure 2.

[*]e-mail: m.mukhlish.ghany.al.fatah@student.tue.nl
[†]e-mail: b.claessens@tue.nl
[‡]e-mail: m.schoukens@tue.nl

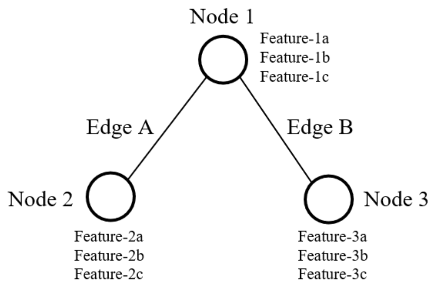


Figure 1: A Graph with 3 nodes and 2 edges

$$X = \begin{bmatrix} Feature-1a & Feature-1b & Feature-1c \\ Feature-2a & Feature-2b & Feature-2c \\ Feature-3a & Feature-3b & Feature-3c \end{bmatrix} \tag{1}$$

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{2}$$

$$D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Euclidean data is data residing in Euclidean space. In mathematics, the Euclidean distance of two points is the length of a line segment between those points. It can be calculated by operating the Pythagorean formula to the two points in a Cartesian coordinate. Text (1D flat shape) and images (2D grid shape) are instances of Euclidean data.

On the other hand, non-Euclidean data are other data types that do not adhere to the axioms and postulates of Euclidean geometry. In non-Euclidean data, it can be illustrated that the shortest path between two points is not necessarily a straight line. In other words, things that are similar to each other are not necessarily close if using Euclidean distance as the metric. Graph data is a non-Euclidean data [6] since the distance between nodes cannot be physically measured by simply using Pythagorean theorem.

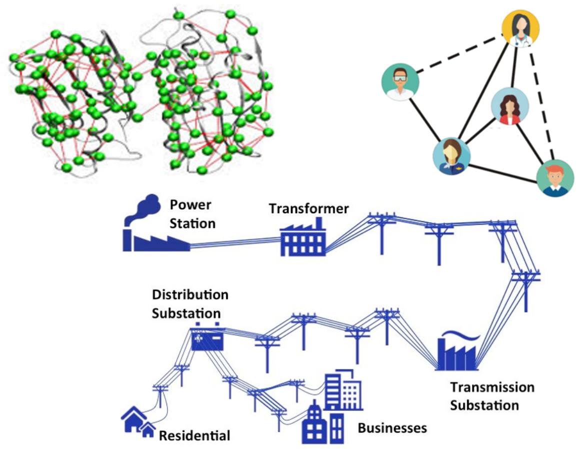


Figure 2: Proteins [30], social networks [1], and electrical power grids [15] are all graphs

Many traditional machine learning frameworks are well proven on Euclidean data. However, the majority of problems being faced by humans are mostly in non-Euclidean space. In fact, many naturally generated data are in graph shape and the problem space is not Euclidean. Therefore, it is of utmost importance to find a generalization of the classical machine learning framework from Euclidean domain to non-Euclidean domain, especially in graphs.

3 Neural Network

3.1 Supervised Machine Learning

Based on the available data, machine learning can be generally boiled down into two types, namely unsupervised machine learning and supervised machine learning. Unsupervised machine learning is a branch of machine learning that only utilizes a set of known inputs data [3]. One of the most well known applications of unsupervised machine learning is data clustering.

Different from the previous type, supervised machine learning makes a predictive model out of a set of known inputs and outputs data [27]. Based on the output type, supervised machine learning can be divided into regression tasks and classification tasks. Regressions are when the outputs are numbers, while classifications are for class-type outputs. For example, a regression task tries to predict the price of houses in the future based on previous trend on calendar time, while a classification task attempts to classify which number a handwritten number image refers to.

3.2 Fully Connected Neural Network

Neural network (NN) is a supervised machine learning framework. Inspired by biological neural networks that shape human brains, NN is a framework that builds an intelligence system learning to perform tasks by observing examples.

NN is constituted by neurons composed in a series of layers connected to one another. One layer is called the input layer which is assigned to store the input data. Another layer is the output layer residing on the opposite end of the network awaiting the result of the process. In between the input and output are the hidden layers which determine how to process the information flowing to the output. The connections between the units are called weights.

A Fully Connected Neural Network (FCNN) is the simplest form of NN [22]. FCNNs are formed by a set of fully connected layers that link each unit in the previous layer to every unit in the next layer. This architecture is depicted in figure 3.

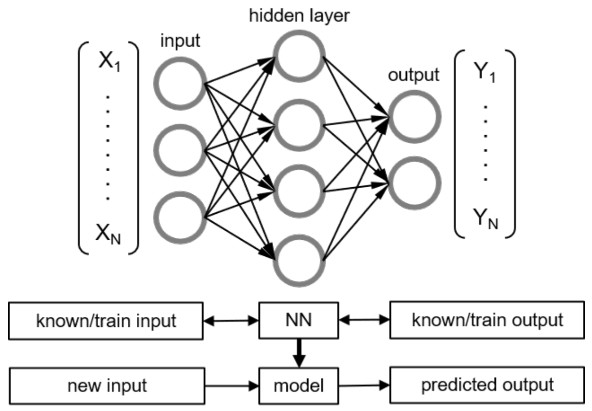


Figure 3: A simple FCNN architecture with one input layer, one hidden layer, and one output layer

The following is a brief process of FCNN training. Each unit obtains inputs from the units to its left, and afterward the values are multiplied by the weights of the connections they follow. These weights are also known as trainable parameters, since these values are the ones that will be tuned in the process. The units are then applied to non-linear functions (e.g. sigmoid or tanh) to extract the hidden features.

The parameter values are firstly guessed to make an initial output. This current output will be compared with the already known output (ground truth). The difference (error/loss) between the predicted output and the ground truth is calculated to check whether the parameters have satisfied the prediction model. If it is yet accurate, the parameters are modified with respect to the derivative of the error to the parameters. This tuning process is repeated for a number of iterations until the model performs well enough. This optimization process can be performed using well-known algorithms such as stochastic gradient descent [26] or ADAM optimizer [17].

Based on the objective, FCNN can either be implemented for classification or regression tasks. Different cases need modification in the model framework. A classification task can use logistic regression error as the loss function, and a regression task may utilize Mean Square Error [24].

FCNN framework is commonly used for flat-shaped data. Different data types such as grid-shaped data in digital image need to be flattened first if it is to be applied in this framework. The major advantage of FCNN is that they are structure agnostic i.e. there are no special assumptions that have to be made for the input. While being structure agnostic makes FCNN widely applicable, such network tends to have worse performance than a special-purpose network designed for a specific structure.

3.3 Convolutional Neural Network

Convolutional Neural Network (CNN) is a more advanced NN framework for a more diverse data type [2]. Convolution is a mathematical function to extract locality features of a data. It can be implemented to multiform data dimensions (flat/one dimensional data, two dimensional data, or higher). A popular convolution application is in digital image processing. A digital Image is a two-dimensional grid-shaped data with fixed size of row and column as well as fixed node ordering. A two-dimensional convolution filter can be applied for instance to detect border features in an image. Figure 4 shows a CNN architecture.

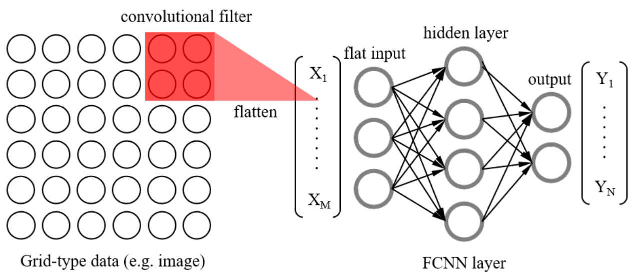


Figure 4: A simple Convolutional Neural Network

A simple CNN architecture in general consists of a sequence of three main layers: convolutional layer, pooling layer (optional), and fully-connected layer [2]. To put it simply, a CNN applies convolutional filters first to a grid-type data to extract the locality features, flatten the convolution output, and afterwards passes it on as an input to a FCNN.

3.4 Graph Neural Network

Modern deep learning techniques are mostly designed for simple sequences (FCNN) and grids (CNN). CNNs can only be run on Euclidean data such as text (1D flat shape) and images (2D grid shape). On the other hand, these data can also be viewed as instances of graphs. For example, every point in a grid data can be regarded as a node in graph data, whereas the surrounding points can be considered as its neighboring nodes. However, it is difficult to define a localized convolutional filter for graphs, which obstructs the transformation of CNN from Euclidean domain to non-Euclidean domain. A way to enable operating convolutional filters to graphs needs to be defined.

Recent advances in Neural Network offer an opportunity to leverage the topology of a graph data. This framework is known as Graph Neural Network (GNN). Protein link prediction in biomedicine and network classification in social networks are two popular GNN applications.

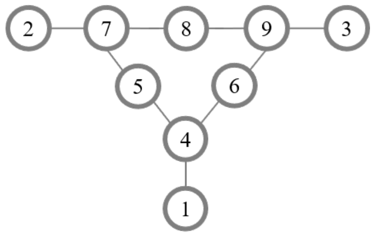


Figure 5: A graph data with 9 nodes

GNN tries to imitate CNN's locality feature extraction to graph data. Take a look at figure 5 showing a graph data with 9 nodes with its edges as connectivity. Applying a straightforward CNN framework to this data will fail since the graph has no fixed node ordering or reference point. Moreover, the graph has arbitrary size with complex topological structure with no spatial locality like grids. GNN solves this problem by using a different mechanism.

3.4.1 Message Passing

The core of GNN is its message passing. Message passing is a mechanism where a target node collects information from its neighbors [20]. This technique ensures that every node senses information from its surroundings. This way, the locality features in every sub location of a graph can be extracted and absorbed by each node. Figure 6 illustrates a message passing happens at node 4, where it obtains information from node 1, 5, 6, and node 4 itself from the previous state/layer L.

$$X_{4,L+1} = \sigma((X_{1,L} + X_{4,L} + X_{5,L} + X_{6,L})W_L) \tag{4}$$

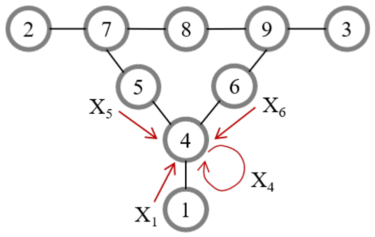


Figure 6: One node's message passing

This message passing at node 4 happens at every node in a graph data for one GNN layer. The formula shown above is the simplest form of message passing with summation operation i.e. only summing the neighboring data. The message passing calculation later on is multiplied by a trainable parameter matrix W that will be trained in the training process.

Message passing mechanism is a way from GNN to mimic CNN's convolutional filter for graph-shaped data. With a similar objective (locality feature extraction), it can be seen as an equivalent convolutional filter for non-Euclidean graph data.

3.4.2 The Simplest GNN: Summation Message Passing

One way to mathematically implement a GNN message passing is using an adjacency matrix. An adjacency matrix is a matrix that represents the topology of a graph data. For example, the matrix at the left side of figure 7 is the adjacency matrix (added by an identity matrix with the same size for diagonal location) for graph in figure 6. The i_{th} row of this matrix represents the neighbors position of the i_{th} node. For instance, the 4th row in the left red box shows the neighbors of node 4, which are node 1, 4, 5, and 6. By doing a matrix multiplication on the adjacency matrix $A+I$ to data matrix X, a simple summation message passing can be made.

Afterwards, the result is multiplied by a trainable parameter matrix W. The row size of W is the same with the column size of matrix X, and the column size of matrix W can be chosen to determine how many features will be generated in the next GNN layer. Finally, the parameterized value is transformed by a non-linearity activation function. In short, that is how one GNN layer is formed.

$$X_{L+1} = \sigma((A+I)X_L W_L) \tag{5}$$

$$\underbrace{\begin{pmatrix} 1&0&0&1&0&0&0&0&0\\ 0&1&0&0&0&0&1&0&0\\ 0&0&1&0&0&0&0&0&1\\ \boxed{1\;\;0\;\;0\;\;1\;\;1\;\;1\;\;0\;\;0\;\;0}\\ 0&0&0&1&1&0&1&0&0\\ 0&0&0&1&0&1&0&0&1\\ 0&1&0&0&1&0&1&1&0\\ 0&0&0&0&0&0&1&1&1\\ 0&0&1&0&0&0&0&1&1 \end{pmatrix}}_{A+I} * \underbrace{\begin{pmatrix} x_1^1 & x_1^2\\ x_2^1 & x_2^2\\ x_3^1 & x_3^2\\ x_4^1 & x_4^2\\ x_5^1 & x_5^2\\ x_6^1 & x_6^2\\ x_7^1 & x_7^2\\ x_8^1 & x_8^2\\ x_9^1 & x_9^2 \end{pmatrix}}_{X_0} * \underbrace{\begin{pmatrix} \text{Trainable parameter, size:}\\ \text{row = features at L}\\ \text{col = features at L+1} \end{pmatrix}}_{W_0}$$

Figure 7: One layer of the simplest GNN formula in matrix notation

3.4.3 Graph Convolutional Network

Graph Convolutional Network (GCN) defines the convolutional operation on the graph domains by operating on spatially close neighbors [18]. The only difference of GCN with a simple summation GNN is that GCN uses an averaging message passing [20]. It can be formed by pre-processing the adjacency matrix with the inverse of the degree matrix. A degree matrix D denotes the number of neighbors of each node. If the inverse of the degree matrix D^{-1} is multiplied to the adjacency matrix $A+I$, the result forms an averaging message passing as shown in figure 8.

$$X_{L+1} = \sigma(D^{-1}(A+I)X_L W_L) \tag{6}$$

$$\underbrace{\begin{pmatrix} 1/2&0&0&1/2&0&0&0&0&0\\ 0&1/2&0&0&0&0&1/2&0&0\\ 0&0&1/2&0&0&0&0&0&1/2\\ \boxed{1/4\;\;0\;\;0\;\;1/4\;\;1/4\;\;1/4\;\;0\;\;0\;\;0}\\ 0&0&0&1/3&1/3&0&1/3&0&0\\ 0&0&0&1/3&0&1/3&0&0&1/3\\ 0&1/4&0&0&1/4&0&1/4&1/4&0\\ 0&0&0&0&0&0&1/3&1/3&1/3\\ 0&0&1/3&0&0&0&0&1/3&1/3 \end{pmatrix}}_{D^{-1}(A+I)} * \underbrace{\begin{pmatrix} x_1^1 & x_2^1\\ x_1^2 & x_2^2\\ x_1^3 & x_2^3\\ x_1^4 & x_2^4\\ x_1^5 & x_2^5\\ x_1^6 & x_2^6\\ x_1^7 & x_2^7\\ x_1^8 & x_2^8\\ x_1^9 & x_2^9 \end{pmatrix}}_{X_0} * \underbrace{\begin{pmatrix} \text{Trainable parameter, size:}\\ \text{row = features at L}\\ \text{col = features at L+1} \end{pmatrix}}_{W_0}$$

Figure 8: One layer of GCN formula in matrix notation

3.4.4 Multi Layers GNN and Hidden Features Extraction

As mentioned before, hidden features extraction can be done by adjusting the size of the parameter matrix W. The number of rows in W is the same with the column size of matrix X i.e. the number of data features, and the column size of matrix W can be set to determine how many features are generated in the next GNN layer.

Figure 9 illustrates two GNN layers (or two hops away GNN) on graph data. Initially, each node has 2 features. After applying the first GNN layer, the data is transformed to have 8 hidden features. The second GNN layer then transforms it back to 2 hidden features. At the last state, all nodes now have two node away message passing, or two hops away information acquiring.

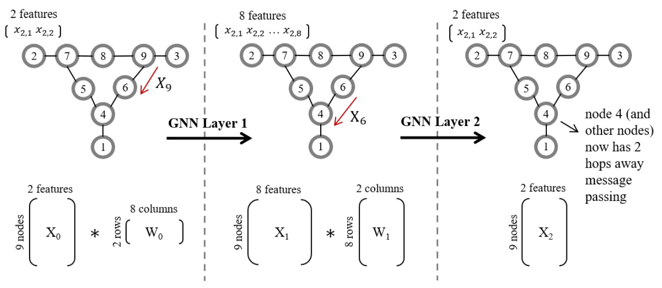


Figure 9: Features extraction in multi layers GNN

3.4.5 GNN as Another Layer in a Deep Neural Network

Similar to CNN, a GNN layer can also be used as a preprocessing layer in a Deep Neural Network architecture. Output of the GNN layer from a graph data can be flattened and passed on to a FCNN layer. In this structure, the GNN layer can be seen as just another layer in the Deep Neural Network as shown in figure 10.

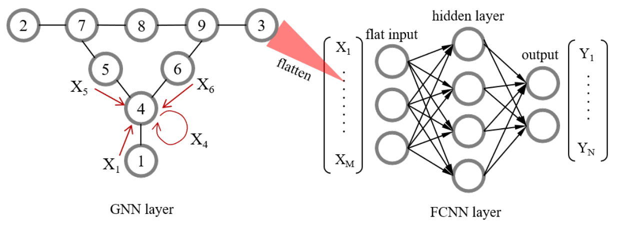


Figure 10: GNN as just another layer in a Deep Neural Network

4 Power Flow

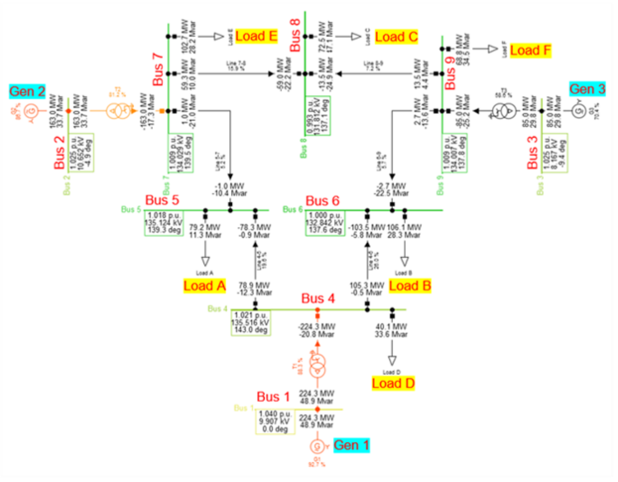


Figure 11: Electrical power grid

Figure 11 illustrates an example of an electrical power grid. Every bus in a power grid can be of type slack bus, PV bus, or PQ bus. At each bus, 4 variables are calculated: active power (P), reactive power (Q), voltage (V), and voltage angle (δ). This calculation is known as power flow or load flow. These variables calculation is important in power system analysis. For instance, a generator needs to know this data to determine how much power it needs to generate to satisfy the demand in the grid [9] [28].

Slack bus is also known as the reference bus. It is used to balance the active power and reactive power in a grid by emitting or absorbing them. There is only one slack bus in a grid. Power flow calculates V and δ of slack bus.

PV bus is also known as generator bus. The P and V are specified for this type of buses. PV buses have constant power generation and voltage that are controlled through a prime mover. Power flow calculates Q and δ of PV bus.

PQ bus is also known as load bus. In PQ buses, the P and Q are specified. Power flow will be used to find the V and δ.

Table 1: Buses type in a power grid

Bus type	Known variables	Unknown variables
Slack bus	V, δ	P, Q
PV bus	P, V	Q, δ
PQ bus	P, Q	V, δ

Power flow calculation tries to find the unknown variables in a bus based on its known variables using electrical formulas. The calculation applies an iterative technique such as the Newton-Raphson method [12]. A more advanced power flow calculation is Optimal Power Flow (OPF) which attempts to find the optimal power that the generators have to produce to satisfy a given demand [7]. The optimality is evaluated with regard to the cost that each generator incurs to generate the required power. Several constraints such as power grid technical limit and cost function are subject in OPF calculation. The case that is investigated in this work is the regular power flow, not the optimal one.

Calculating the power flow using electrical data fully in AC form is known as AC Power Flow (ACPF). Since AC values involve non-linear (sinusoidal) data, it relatively takes a longer time for the iterative method to finish the calculation, while electrical providers in the field need to finish the calculation quickly (around every 5 minutes [28]) for field related problems.

A faster method to solve power flow calculation is using DC power flow (DCPF). DCPF uses several assumptions that simplify AC power flow's non-linear complexity [9]: it assumes that the lines voltages are flat 1 p.u. at all buses and voltage angle differences between neighboring buses are small so that $\cos(\delta_1 - \delta_2) = 1$ and $\sin(\delta_1 - \delta_2) = \delta_1 - \delta_2$. As a simplifying method, DCPF is not as accurate as ACPF but runs faster. However, the small angle approach fails for heavily-loaded networks, as the voltage angle differences become large [8]. Electrical providers are able to use DCPF as a warm start up method to start the iterative ACPF for a complementary solution [11].

A hypothetically faster method is using a machine learning model for power flow data prediction. NN based models can be utilized to predict the unknown variables of power flow by harnessing the known variables to make a predictive model. Even better, GNN models can improve the model performance by also making use of the power grid topology information.

5 Experiment: GNN for Power Flow

5.1 Introduction

In this study, the opportunity to implement GNN on a graph-shaped electrical power grid is the main thing to seek, specifically in power flow application. This study investigates how GNN models can make a more powerful prediction by not only utilizing the unknown variables of power flow, but also leveraging the power grid connectivity information. Based on this case, the task would be a regression.

There are several assumptions to consider in this study. Firstly, the experiment only focuses on PQ buses in the grid to simplify the case. Mixing PQ buses alongside slack and PV buses would complicate the pattern recognition process in the training phase. This study primarily wants to find the pattern of the model performance albeit it is not in the most ideal case. The investigated power grid consists of 14 buses as shown in figure 12.

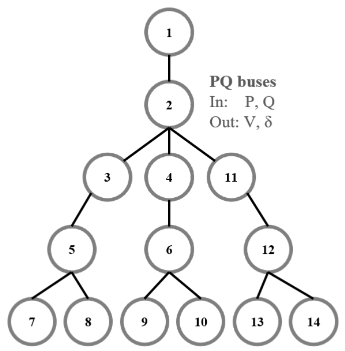


Figure 12: The investigated 14 bus power grid

Secondly, the type of GNN framework used in this study is GCN. GCN is considered as the basis of GNN and has been well proven on various GNN applications. The GNN model in this study will only utilize the features of the buses/nodes in the graph data i.e. the P and the Q variables. Features of the lines/edges such as electrical current will be disregarded to simplify the case. In the dataset generation, the type and length of the lines between the buses in the simulated power grid are designed to be exactly the same.

Thirdly, this study tries to compare the result of FCNN versus GNN model on graph data. Furthermore, the performance of one hops away and two hops away GNN model will be compared. The elaboration will be discussed in the next section.

Lastly, this study also compares several cases where there are loops in the graph data. The cases are represented in figure 13. The grid shapes were made so that there are small, medium, and large sized loops in the graph data. This setup will test how the GNN model would react to such looped graph cases.

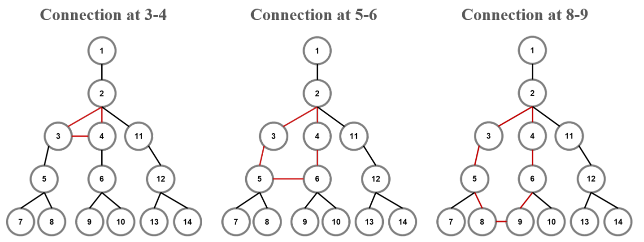
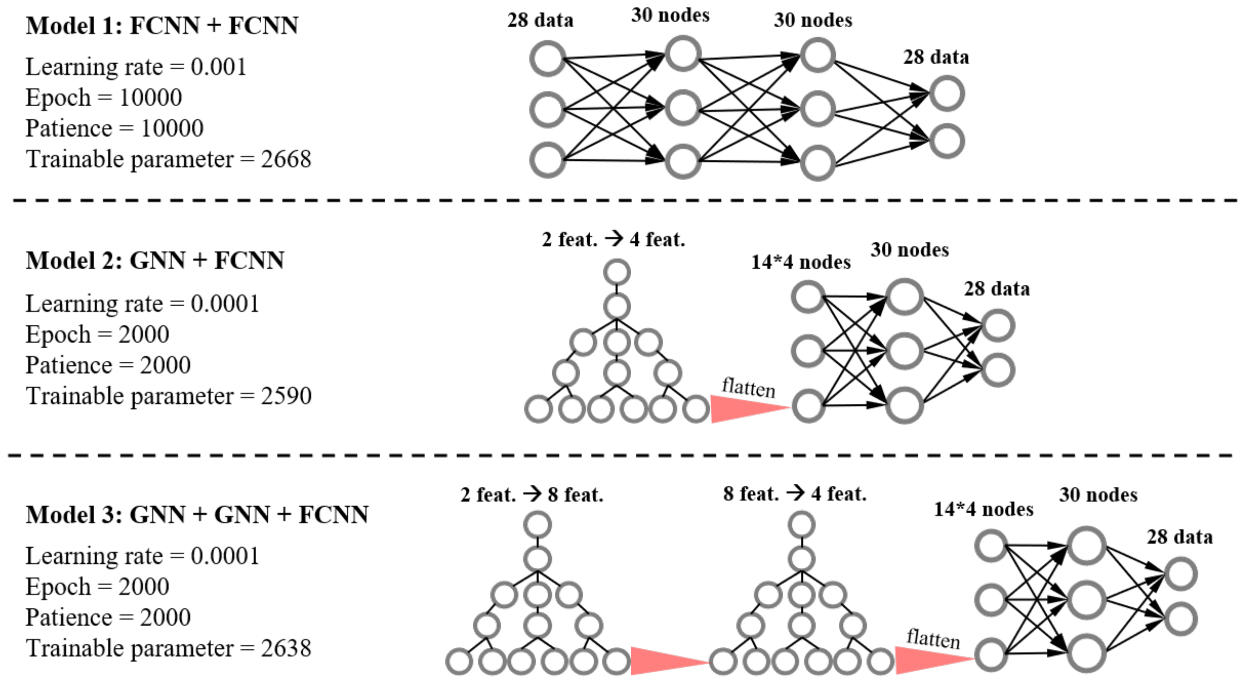


Figure 13: The 14 bus power grid with various sized loops

5.2 Model Architecture

There are 3 model architectures that will be compared in this study. Model 1 is a two layers FCNN, model 2 is a combination of one layer GNN and FCNN, and model 3 is a two hops away GNN combined with one layer FCNN. The model architectures can be seen in figure 14. These architectures has a certain number of layers and nodes that were selected based on several reasonings.

First of all, the three models were made so that the total number of trainable parameters are roughly the same. The aim of this comparison is to evaluate which model has the best performance. To do so, the models have to be designed with the same level of complexity. The number of layers and the total number of parameters are made as close as possible so all models can be compared equivalently. The calculation of the model parameter is shown in table 2.



Model 1: FCNN + FCNN

Learning rate = 0.001
Epoch = 10000
Patience = 10000
Trainable parameter = 2668

Model 2: GNN + FCNN

Learning rate = 0.0001
Epoch = 2000
Patience = 2000
Trainable parameter = 2590

Model 3: GNN + GNN + FCNN

Learning rate = 0.0001
Epoch = 2000
Patience = 2000
Trainable parameter = 2638

Figure 14: The three models to be compared in the experiment

Table 2: The three models' parameters calculation

Model 1				Model 2				Model 3			
param	row	col	total	param	row	col	total	param	row	col	total
w1	30	28	840	w1	2	4	8	w1	2	8	16
b1	30	1	30	b1	4	1	4	b1	8	1	8
w2	30	30	900	w2	30	56	1680	w2	8	4	32
b2	30	1	30	b2	30	1	30	b2	4	1	4
w3	28	30	840	w3	28	30	840	w3	30	56	1680
b3	28	1	28	b3	28	1	28	b3	30	1	30
Total parameter			2668	Total parameter			2590	w4	28	30	840
								b4	28	1	28
								Total parameter			2638

For the number of hidden layers and nodes, these numbers were selected based on a simple-picky grid search for hyperparameter tuning. The reason was because the cost of a complete grid search method on the GNN framework was too expensive in time consumed. Although perhaps they were not the best, the selected model frameworks were enough to generate a visible pattern in the result to draw a conclusion.

The complete hyperparameter list can be seen in figure 14. Every hidden node in the models uses tanh activation function, except the nodes in the last layer. The models employ the ADAM optimizer. In the training phase, all data is normalized using mean and standard deviation normalization to ease the gradient descent process [16].

5.3 Dataset Generation

The power flow dataset is generated using Power Factory 2020 power system simulation [4]. For this dataset generation, the program iteratively runs power flow calculation. For each iteration, the value of loads at PQ buses are varied unique-randomly by up to 50% to make a unique and random data characteristic. Main variables (P, Q, V, and δ) at each bus are collected and saved to a spreadsheet file.

The data structure for 1 dataset can be seen in table 3. One dataset consists of 2000 data points. Each datapoint contains data of 1 power flow calculation from the simulation. Therefore, there are 2000 unique power flow data calculations in 1 dataset. The program was simulated to generate 102 datasets: 1 dataset for train dataset, 1 dataset for validation dataset, and 100 datasets for test dataset. If this dataset is intended to be implemented on a FCNN model, it has to be flattened first.

Table 3: One power flow dataset

Data point	**Node**	**Input**		**Output**	
Data point 1	1	P_1	Q_1	V_1	δ_1
	2	P_2	Q_2	V_2	δ_2
	3	P_3	Q_3	V_3	δ_3
	...	...	...	...	...
	14	P_{14}	Q_{14}	V_{14}	δ_{14}
...	...				
Data point 2000	1	P_1	Q_1	V_1	δ_1
	2	P_2	Q_2	V_2	δ_2
	3	P_3	Q_3	V_3	δ_3
	...	...	...	...	...
	14	P_{14}	Q_{14}	V_{14}	δ_{14}

5.4 Experiment Steps

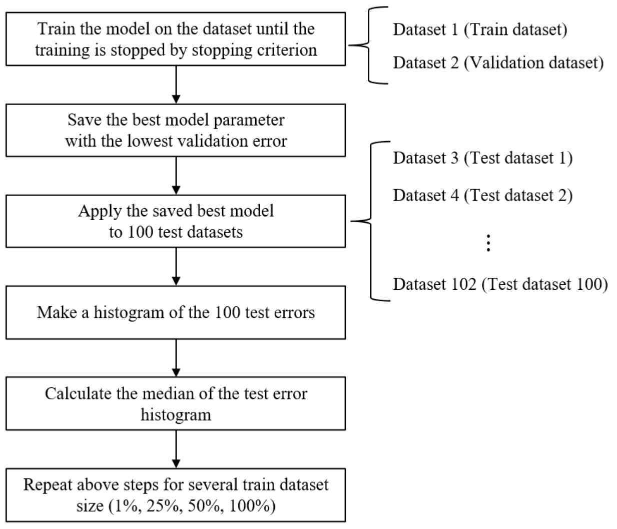

Train the model on the dataset until the training is stopped by stopping criterion
Dataset 1 (Train dataset)
Dataset 2 (Validation dataset)
Save the best model parameter with the lowest validation error
Apply the saved best model to 100 test datasets
Dataset 3 (Test dataset 1)
Dataset 4 (Test dataset 2)
⋮
Dataset 102 (Test dataset 100)
Make a histogram of the 100 test errors
Calculate the median of the test error histogram
Repeat above steps for several train dataset size (1%, 25%, 50%, 100%)

Figure 15: The experiment steps

The steps of the experiment are as follows. First, train the model on train and validation dataset. Record the train and validation error for every iteration. The training process is stopped based upon the stopping criterion i.e. maximum number of epoch/train iteration and patience. Save the best model parameter i.e. parameter yielding the lowest validation error. Apply the saved best model parameter to 100 test datasets. Plot the histogram of the 100 test errors.

There are 2 types of error calculation used in this experiment, namely Mean Square Error (MSE) and Normalized Root Mean Squared Error (NRMSE). MSE is a loss function normally used for regression tasks. MSE calculates real error between the predicted output and the ground truth. MSE punishes a higher error more severely because the loss will be squared, making the system detect the loss better [24]. NRMSE is RMSE that is divided by the variance of the model prediction. NRMSE is used to ensure that the predicted values are indeed bonafide and not just merely data averaging.

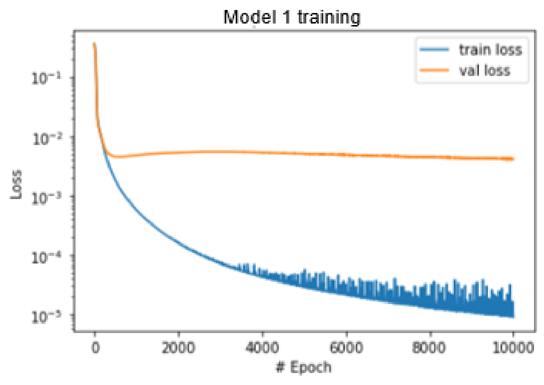
Next, calculate either mean or median of the test error histogram. Median is more commonly preferred over the mean when the frequency distribution of the data is not balanced [31]. If the data is in normal distribution, the mean, median, and mode are identical. However, if the data is skewed, the mean will fail to provide the best central location for the data, since the skewed data drags the mean away from the typical value. The median is more likely to retain this position as it is not as strongly influenced by the skewed trend. The median of the test error histogram can be taken as a reliable indicator for the model performance since it has been tested over large test datasets.

Lastly, repeat all the steps over again for several different train dataset sizes: 1% (20 data points), 25% (500 data points), 50% (1000 data points), and 100% (2000 data points, default size). The complete steps are illustrated in figure 15.

6 Result and Analysis

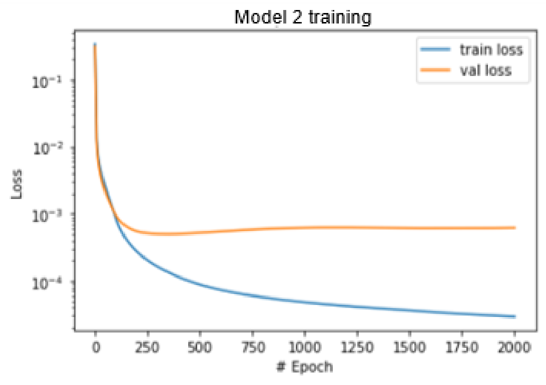
6.1 14 Bus Power Grid

The result of the experiment for the 14 bus power grid can be seen in the following section. Figure 16, 17, and 18 show the training process of model 1, 2, and 3 for the case of 25% train dataset size. The train loss and validation loss for every training iteration are recorded in the blue and orange lines. The best model saved on each training case is the one yielding the lowest validation loss. Take a look that the lowest validation loss is not always located at the highest training iteration or the lowest train loss.



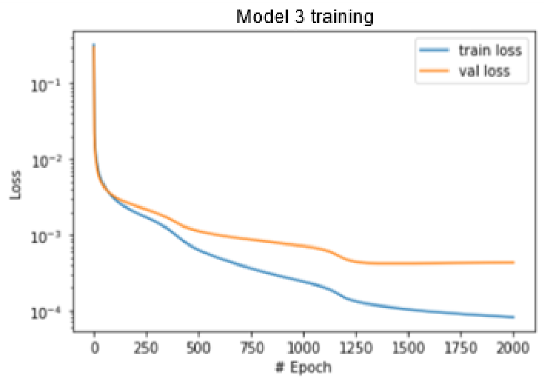
```
last epoch: 10000, train loss: 0.0000090, val loss: 0.0040871
best epoch: 9926, train loss: 0.0000181, val loss: 0.0040692
```

Figure 16: The model 1 training process for 25% train dataset size



```
last epoch: 2000, train loss: 0.0000292, val loss: 0.0006154
best epoch: 344, train loss: 0.0001363, val loss: 0.0004968
```

Figure 17: The model 2 training process for 25% train dataset size



```
last epoch: 2000, train loss: 0.0000807, val loss: 0.0004249
best epoch: 1421, train loss: 0.0001087, val loss: 0.0004132
```

Figure 18: The model 3 training process for 25% train dataset size

After the best models are acquired, they are tested over 100 test datasets. The histogram of those 100 test errors are plotted to see the distribution of the error values. However, the Probability Density Function (PDF) is used to simplify the visualization instead of histogram, while they still have the same objectives. PDF illustrates the probability of the value in x axis presented in the graph, with the total area inside the PDF is one.

The distribution of MSE and NRMSE for the 100 test errors are

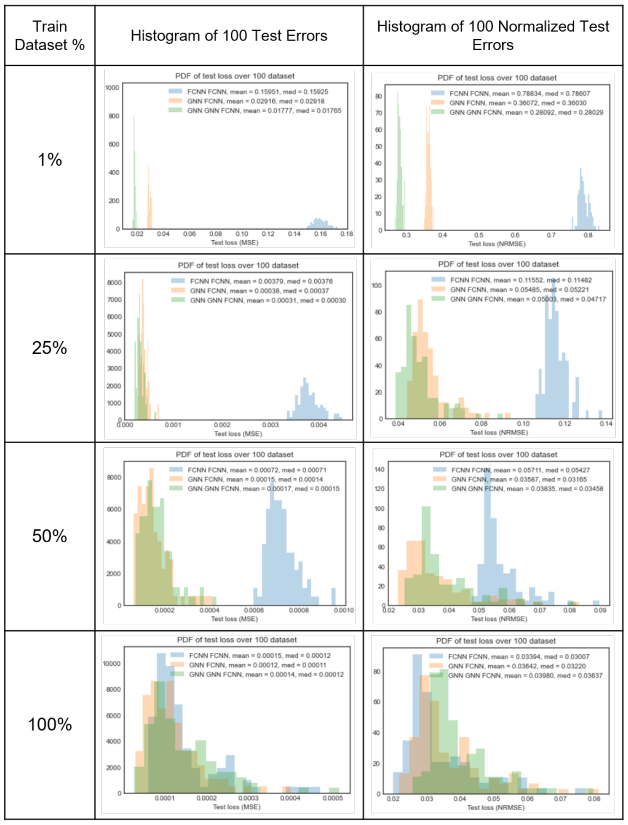


Figure 19: The histograms of the three models on 1%, 25%, 50%, and 100% train dataset. Different rows have different x axis scales.

plotted in figure 19. It can be seen that both error types are identical in pattern. Moreover, the NRMSE plots are all below the value of one in x axis, meaning that the model prediction is indeed credible. Notice that different rows in figure 19 have a different scale of x axis. A higher train dataset size makes the gap between GNN and FCNN histogram smaller, and the body of the histogram gradually moves to the left side. It indicates that increasing the number of train dataset generates a lower test error and a more accurate model.

Afterwards, the median of every histogram is plotted in one graph so that the test error trend line can be seen and directly compared in one axis, as illustrated in figure 20 and 21. Notice that the y axis is served in logarithmic scale, while the x axis in a linear scale.

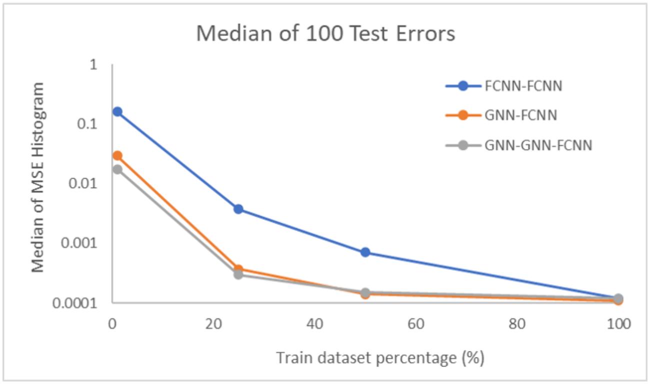


Figure 20: The medians of MSE histograms of 14 bus grid

The result shows that for a same model complexity i.e. same number of trainable parameters, GNN-based models outperform FCNN

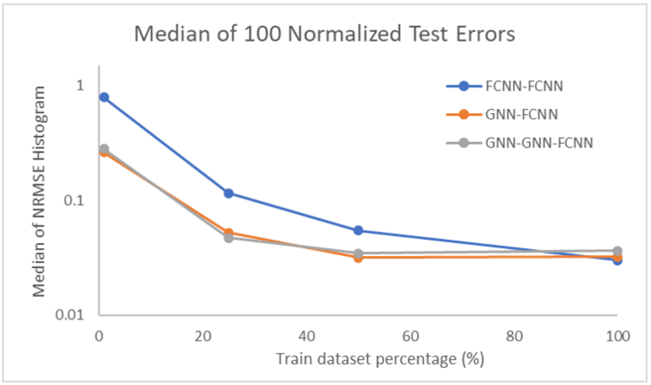


Figure 21: The medians of NRMSE histograms of 14 bus grid

especially in lesser train dataset. This can be said in other words that a GNN model needs less parameters to match the performance of a FCNN model. This trait exists because a GNN layer utilizes parameter sharing.

Take a look at figure 22. A FCNN layer uses more parameters because 1 parameter only connects 1 node in a layer to another node in the next layer. For 18 flat input and 36 flat output, a FCNN layer requires 18x36 = 648 parameters.

This is not the case in a GNN layer. In GNN, a single column parameter in matrix W is jointly used by all nodes (rows) in graph data matrix X, as shown in figure 23. For an input with size of 18 data (9 node x 2 features in graph representation) and output of 36 data (flat), a GNN layer only needs 2x4 = 8 parameters.

This parameter sharing can be established because aside from utilizing the data, a GNN model also leverages the data topology information in the form of adjacency matrix. This data connectivity knowledge makes the GNN model to be kind of 'aware' of the model and data structure, unlike the FCNN model that is structure-agnostic. Therefore, when a new unknown data comes to a GNN model, the model is already conscious of the data inter-dependency and trains the model from a more advanced starting point than FCNN. This trait makes GNN become better in generalizing the inter-nodes relationship in graph data and thus reducing overfitting problems. It also makes the GNN training process not needing many iterations compared to FCNN.

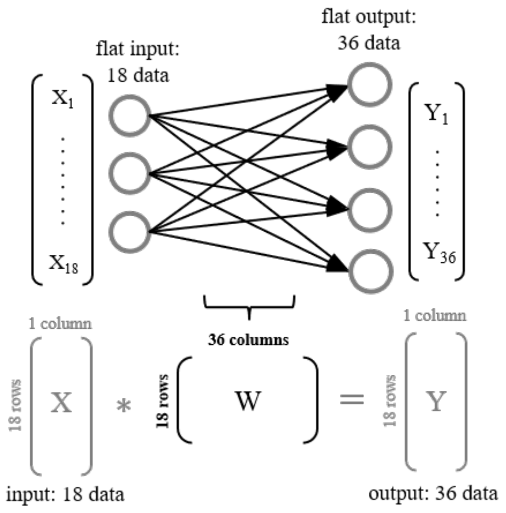


Figure 22: A FCNN layer needs many parameters

One situation where this property makes GNN more superior than

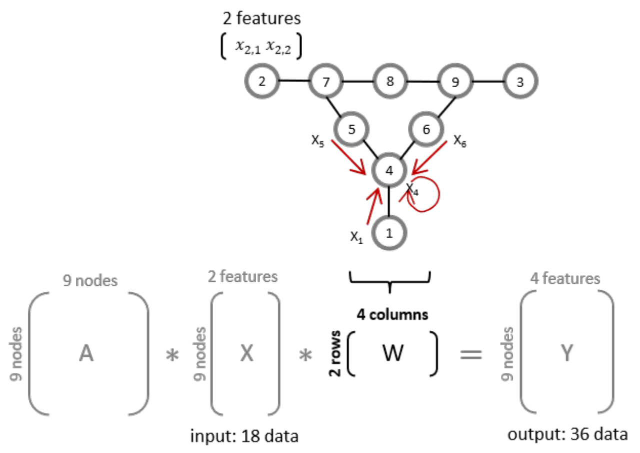


Figure 23: A GNN layer needs less parameters

a traditional FCNN is in a case when the data is slightly available, for instance when the cost of acquiring new data is expensive. Having the same small available data, GNN is able to make the prediction model more accurate by utilizing the connectivity information of the data. This case is illustrated in figure 20 for small-sized train dataset, where GNN-based models generate considerably lower test errors than FCNN models.

As for the comparison between one hop away and two hops away GNN, there is hardly any visible pattern on the result in figure 20 and 21. Although the median of the test errors of the model 3 (grey line) at figure 20 seems to be a little bit lower than that of model 2 (orange line) in smaller train dataset size, perhaps it is too early to make any conclusion from this result alone.

Theoretically, applying more GNN layers will extract a deeper hidden features that leads to a more powerful model. However, utilizing a too deep GNN architecture or too many GNN layers hypothetically would not be beneficial and instead will harm the uniqueness of the specific-local feature of a graph data [20]. An additional experiment is performed to determine the results of applying such case. The yellow line at figure 24 indicates the result of four hops away GNN.

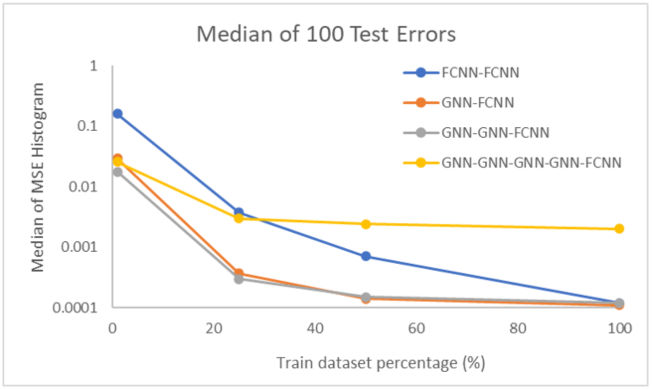


Figure 24: The medians of MSE histograms of 14 bus grid with four hops away GNN depth

Using GNN message passing repeatedly over a graph data will make the features of the nodes become more and more identical with the far neighbors. This setup leads to an overgeneralized model. There have to be a certain number of maximum GNN layers that can be optimally applied to a graph data. This number will depend heavily on the size and the depth of the graph.

6.2 14 Bus Power Grid with Loop

6.2.1 The Problem with GNN on Looped Graphs

The experiment for the 14 bus power grid with loop entangles similar steps with the previous section. First of all, the investigated looped-power grids were made so that the loops represent various sizes. The grid with connection at bus 3-4, at bus 5-6, and at bus 8-9 respectively has small, medium, and big sized loops. It was logical to expect that there will be a pattern drawn in the test error histogram. The result is shown in figure 25, 26, and 27.

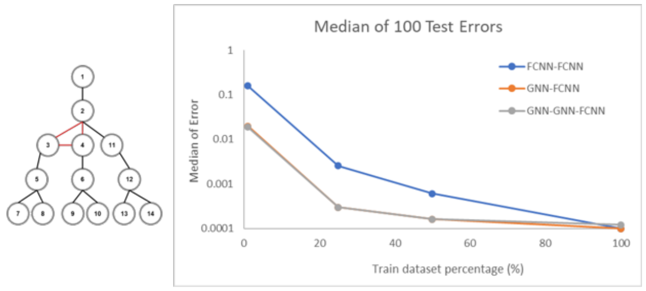


Figure 25: The medians of MSE histograms of grid with loop at 3-4

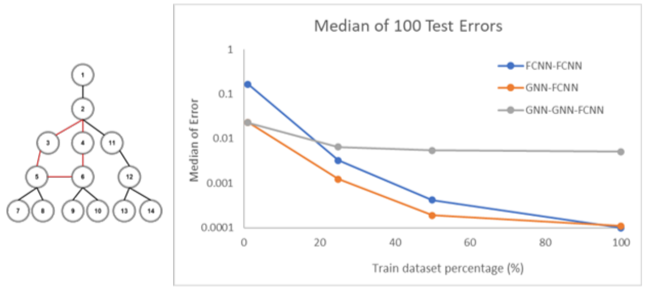


Figure 26: The medians of MSE histograms of grid with loop at 5-6

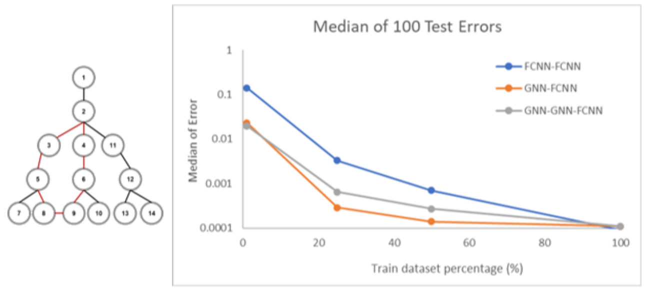


Figure 27: The medians of MSE histograms of grid with loop at 8-9

Looking at this result, it can be seen that there is a problem with the 2 hops away GNN model on the power grid with connection at bus 5-6 (medium sized loop). The gradient descent seems to be not working well. The model error is not improved starting at a certain point even though the train dataset is increased. The cause can be several things. One of them is that the model was stuck at a local minima in the gradient descent process during the training. There are several ways to check this issue.

6.2.2 Making the Second GNN Layer Transparent

One way to check whether the model 3 (GNN-GNN-FCNN) is stuck at a local minima is by making the second GNN layer to be transparent. A transparent layer makes sure that the data is not modified by that layer, only bypassed from the previous layer to the next layer. With this setup, the model 3 architecture becomes

GNN-transparent GNN layer-FCNN. Theoretically, this architecture delivers the exact same result with model 2 (GNN-FCNN). When this condition happens, the second GNN layer can be trained alone (by freezing the other layers' parameters) from that point and see whether the model improves and escapes the local minima. However, there is a problem when trying to make a transparent GNN layer.

Take a look at how a FCNN layer is made transparent. The formula of forward propagation in one FCNN layer is shown below.

$$X_{L+1} = \sigma(X_L W_L) \tag{7}$$

To make a transparent layer, the trainable parameter matrix W_L has to be made so that the data matrix at the next layer X_{L+1} is the same with the data matrix at the current layer X_L. The σ is an element-wise non-linear matrix operator that is applied to every element in the matrix, so it can be disregarded in the equation.

$$X_L W_L = X_L \tag{8}$$

$$X_L^{-1} X_L W_L = X_L^{-1} X_L \tag{9}$$

$$W_L = I \tag{10}$$

It can be concluded that making a transparent layer in FCNN can be done by simply assigning the trainable parameter matrix W with an identity matrix I. The case is different for a GNN layer. The formula of one GNN forward propagation is shown in the following.

$$X_{L+1} = \sigma(D^{-1}(A+I)X_L W_L) \tag{11}$$

Same with before, find the value of matrix W_L that makes X_{L+1} equal to X_L to make a transparent GNN layer.

$$D^{-1}(A+I)X_L W_L = X_L \tag{12}$$

$$DD^{-1}(A+I)X_L W_L = DX_L \tag{13}$$

$$(A+I)^{-1}(A+I)X_L W_L = (A+I)^{-1}DX_L \tag{14}$$

$$X_L^{-1} X_L W_L = X_L^{-1}(A+I)^{-1}DX_L \tag{15}$$

$$W_L = X_L^{-1}(A+I)^{-1}DX_L \tag{16}$$

There are several problems with the W_L formula above. First, the equation involves the data matrix X that makes the formula become data dependent. Second, finding the inverse matrix of X is difficult if the matrix X is not square-sized (and in most cases it is not), since a non-square matrix does not have an inverse. Therefore, making a transparent layer out of GNN is not possible.

6.2.3 Applying ResNet to the Second GNN Layer

Another method to get the objective is by applying Residual Network (ResNet) framework on the two hops away GNN model. ResNet is developed to solve the problem of vanishing gradients in a deep learning framework [14]. The architecture of the GNN-GNN-FCNN model with ResNet at the second GNN layer is indicated in figure 28. The resulting test error histograms from this model architecture are depicted in figure 29, 30, and 31.

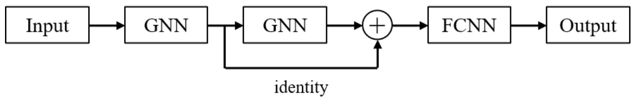


Figure 28: The model 3 architecture with a residual network

The two hops away GNN model is indeed improved when the models employ the ResNet layer. The model 3 error eventually gets lower than the error of model 2. However, there is still barely any pattern visible in the test error histograms with respect to the loop size of the graph. From this result alone, it is still hard to find any conclusion regarding how the presence of loops in a graph affects the performance of GNN.

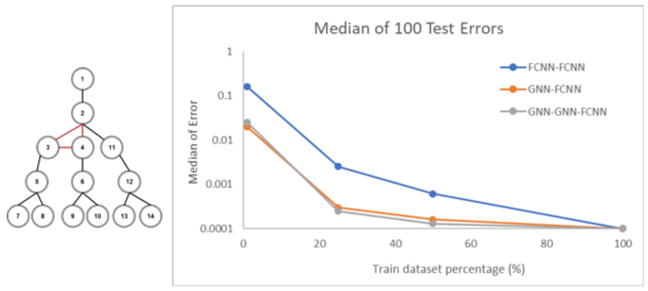


Figure 29: 14 bus grid with loop at 3-4 with ResNet

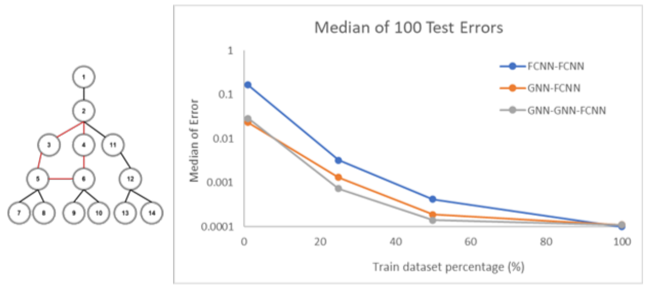


Figure 30: 14 bus grid with loop at 5-6 with ResNet

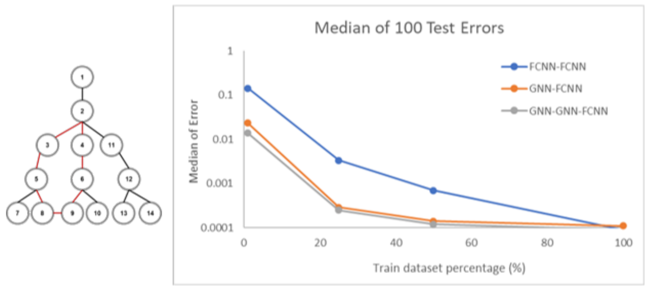


Figure 31: 14 bus grid with loop at 8-9 with ResNet

7 Conclusion

This study provides a comprehensive review of GNN for power flow application. Several conclusions can be observed from the result.

GNN models can leverage a graph's connectivity information by involving adjacency matrix in its operation. With this trait, GNNs utilize parameter sharing that improves model accuracy especially for lower train dataset size. GNN outperforms traditional FCNN in a case when the data is slightly available, for instance when the cost of acquiring new data is expensive.

However, there are several problems with GNN that need to be studied. There is a certain graph topology where GNN faces a problem, namely when the graph contains loops. A further study needs to examine in which condition this issue presents. Furthermore, it is still difficult to make a transparent layer out of GNN, which perhaps will be beneficial for future development.

Lastly, the experiment with looped graphs is yet to make any reliable conclusion. There were no visible patterns from the result of this case. A good starting point can be tried by implementing a two or three hops away GNN model on a graph data containing many nodes and deep connectivity. This case may reveal the benefit of implementing a deep GNN framework in a deep graph data.

8 Recommendation

Several recommendations for further works are presented in the following.

First, a further study shall try other types of GNN frameworks such as Graph Attention Network (GAT) [29] which is claimed to be more powerful than GCN.

Second, the option to utilize the lines/edges features such as electrical current in the graph data can be a helpful addition.

Third, a study needs to check whether GNN is indeed not possible to make a transparent layer. This case will perhaps be needed for future development.

Fourth, an implementation of GNN on a big and deep graph i.e. graph with many nodes and long data connectivity can be a good starting point to check the effect of multi hops away GNN layer on such structure. This case may reveal the benefit of implementing a deep GNN framework on a deep graph data.

Acknowledgments

This work was supported in part by a grant from Lembaga Pengelola Dana Pendidikan (LPDP) Indonesia.

References

[1] I. Ahmad, M. Akhtar, S. Noor, and A. Shahnaz. Missing link prediction using common neighbor and centrality based parameterized algorithm. *Scientific Reports*, 10:364, 01 2020. doi: 10.1038/s41598-019-57304-y

[2] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.

[3] M. Alloghani, D. Al-Jumeily Obe, J. Mustafina, A. Hussain, and A. Aljaaf. *A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science*, pp. 3–21. 01 2020. doi: 10.1007/978-3-030-22475-2_1

[4] V. Astapov, I. Palu, and T. Vaimann. The use of digsilent power factory simulator for "introduction into power systems" lectures. *Electrical, Control and Communication Engineering*, 14:95–99, 12 2018.

[5] D. Bienstock and A. Verma. Strong np-hardness of ac power flows feasibility. *Operations Research Letters*, 47(6):494–501, Nov. 2019.

[6] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *CoRR*, abs/1611.08097, 2016.

[7] M. B. Cain, R. P. O'Neill, , and A. Castillo. History of optimal power flow and formulations. *Federal Energy Regulatory Commission, Increasing Efficiency through Improved Software*, p. 1–31, Dec. 2012.

[8] S. Chatzivasileiadis. Lecture notes on optimal power flow opf. *arXiv:1811.00943v1 [cs.SY]*, Nov. 2018.

[9] K. V. den Bergh, E. Delarue, and W. D'haeseleer. Dc power ow in unit commitment models. KU Leuven, May 2014.

[10] F. Diehl. Applying graph neural networks on heterogeneous nodes and edge features. *33rd Conference on Neural Information Processing Systems*, 2019.

[11] F. Diehl. Warm-starting ac optimal power flow with graph neural networks. 2019.

[12] A. Eltamaly, A. A. Elghaffar, Y. Mohamed, and A.-H. Ahmed. Optimum power flow analysis by newton raphson method, a case study. 12 2018.

[13] N. Guha, Z. Wang, and A. Majumdar. Machine learning for ac optimal power flow. *36th Int. Conf. Mach. Learning*, June 2019.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[15] S. Hosein, P. Hosein, W. Kattick, and V. Ratan. Web application for power grid fault management. pp. 1–5, 08 2016. doi: 10.1109/ICIAS.2016.7824052

[16] L. Huang, J. Qin, Y. Zhou, F. Zhu, L. Liu, and L. Shao. Normalization techniques in training dnns: Methodology, analysis and application. *CoRR*, abs/2009.12836, 2020.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[18] T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *Fifth International Conference on Learning Representations*, July 2018.

[19] B. C. Lesieutre and I. A. Hiskens. Convexity of the set of feasible injections and revenue adequacy in ftr markets. *IEEE Trans. Power Syst.*, 20(4):494–501, Nov. 2005.

[20] J. Leskovic. Cs224w machine learning with graphs, 2019. [Online; accessed on February 27, 2020].

[21] W. Liao, B. Bak-Jensen, J. R. Pillai, Y. Wang, and Y. Wang. A review of graph neural networks and their applications in power systems. Jan. 2021.

[22] P. Marius, V. Balas, L. Perescu-Popescu, and N. Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8, 07 2009.

[23] D. K. Molzahn and I. A. Hiskens. Convex relaxations of optimal power flow problems: An illustrative example. *IEEE Trans. Power Syst.*, 63(5):650–660, May 2016.

[24] A. Ng. Machine learning by stanford university, 2016. [Online; accessed on August 11, 2020].

[25] D. Owerko, F. Gama, and A. Ribeiro. Optimal power flow using graph neural networks. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5930–5934, 2020.

[26] S. Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016.

[27] A. Singh, N. Thakur, and A. Sharma. A review of supervised machine learning algorithms. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1310–1315, 2016.

[28] J. Tong and H. Ni. Look-ahead multi-time frame generator control and dispatch method in pjm real time operations. pp. 1–1, 07 2011.

[29] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *Sixth International Conference on Learning Representations*, July 2018.

[30] M. S. Vijayabaskar, V. Niranjan, and S. Vishveshwara. Graprostr - graphs of protein structures: A tool for constructing the graphs and generating graph parameters for protein structures. *Open Bioinformatics Journal*, 5:53–58, Feb. 2011.

[31] P. von Hippel. Mean, median, and skew: Correcting a textbook rule. *Journal of Statistics Education*, 13:965–971, 07 2005. doi: 10.1080/10691898.2005.11910556

[32] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *IEEE Transactions on Visualization and Computer Graphics*, July 2019.